

1. คาบที่ 7 :ทฤษฎีสำคัญของ Git ,Pointer (6 ชม.)

1.1. บรรยาย (3 ชม.)

1.1.1.ทฤษฎีสำคัญของ Git

Git เป็นระบบควบคุมเวอร์ชัน (Version Control System) แบบกระจาย (Distributed) ที่พัฒนาโดย Linus Torvalds เพื่อช่วยในการจัดการและติดตามการเปลี่ยนแปลงของไฟล์ในโปรเจกต์ โดยเฉพาะในโปรเจกต์ที่มีการพัฒนาซอฟต์แวร์ร่วมกัน Git ได้รับความนิยมสูงเนื่องจากความยืดหยุ่นและประสิทธิภาพในการจัดการโค้ดที่ซับซ้อน

1.1.1.1. Version Control System (VCS)

Git เป็นระบบควบคุมเวอร์ชันที่ช่วยติดตามการเปลี่ยนแปลงของไฟล์ในโครงการ:

- 1) เก็บประวัติการเปลี่ยนแปลง (History)
- 2) สามารถย้อนกลับไปยังสถานะก่อนหน้าได้ (Revert Changes)
- 3) ช่วยให้หลายคนทำงานร่วมกันบนโค้ดเดียวกันได้โดยไม่มีปัญหาเรื่องการชนกันของโค้ด (Merge Conflict)

1.1.1.2. Distributed Version Control

- 1) Git แตกต่างจาก VCS แบบเดิม (เช่น SVN) เพราะทุกคนที่ใช้ Git จะมีสำเนาของ Repository ทั้งหมดในเครื่องตัวเอง
- 2) การทำงานไม่ต้องพึ่งพาเซิร์ฟเวอร์กลาง (Central Server) เสมอไป สามารถทำงานแบบออฟไลน์ได้

1.1.1.3. โครงสร้างพื้นฐานของ Git

Git ใช้โครงสร้างข้อมูลแบบ DAG (Directed Acyclic Graph) เพื่อจัดการ Commit และการเปลี่ยนแปลง:

- 1) **Blob:** เก็บข้อมูลไฟล์
- 2) **Tree:** เก็บโครงสร้างของไฟล์และไดเรกทอรี
- 3) **Commit:** จุดที่บันทึกสถานะของโปรเจกต์

1.1.1.4. คำสั่งพื้นฐาน

- 1) git init: สร้าง Git repository
- 2) git clone: คัดลอก repository จากระยะไกล
- 3) git add: เพิ่มไฟล์เข้าสู่ staging area
- 4) git commit: บันทึกการเปลี่ยนแปลงใน repository
- 5) git push: ส่งการเปลี่ยนแปลงไปยัง repository ระยะไกล
- 6) git pull: ดึงการเปลี่ยนแปลงจาก repository ระยะไกล
- 7) git merge: รวมการเปลี่ยนแปลงจาก branch อื่น
- 8) git branch: สร้างหรือลบ branch
- 9) git checkout หรือ git switch: เปลี่ยน branch หรือย้อนกลับไปยัง commit ก่อนหน้า

1.1.1.5. Branching and Merging

- 1) **Branch:** ช่วยให้งานแยกกันเป็นอิสระ เช่น การพัฒนา Feature หรือแก้ไข Bug
- 2) **Merge:** รวมการเปลี่ยนแปลงจาก Branch อื่นเข้ามา
- 3) หากมีความขัดแย้ง (Conflict) Git จะแจ้งให้ผู้ใช้แก้ไขก่อน

1.1.1.6. Stages in Git

Git มี 3 สถานะหลัก:

- 1) **Working Directory:** ไฟล์ในโฟลเดอร์โปรเจกต์ปัจจุบัน
- 2) **Staging Area:** ไฟล์ที่เตรียมจะ commit
- 3) **Repository:** ไฟล์ที่ถูกบันทึกในระบบ Git

1.1.1.7. การจัดการการเปลี่ยนแปลง

- 1) **Snapshot:** Git บันทึก snapshot ของไฟล์ในแต่ละ commit แทนการบันทึกความแตกต่างของไฟล์ (difference-based)
- 2) ใช้ SHA-1 (Secure Hash Algorithm) เพื่อระบุ commit แต่ละตัว

1.1.1.8. การทำงานร่วมกับ Repository ระยะไกล

- 1) Remote Repository เช่น GitHub, GitLab หรือ Bitbucket เป็นที่เก็บโค้ดที่ทุกคนสามารถเข้าถึงได้

- 2) การดึง (pull) และการส่ง (push) การเปลี่ยนแปลงช่วยให้ทีมทำงานร่วมกันได้อย่างราบรื่น

1.1.1.9. การแก้ไขข้อผิดพลาด

- 1) git reset: ยกเลิกการเปลี่ยนแปลงใน staging area หรือ commit
- 2) git revert: ย้อนกลับการเปลี่ยนแปลงโดยสร้าง commit ใหม่
- 3) git stash: เก็บการเปลี่ยนแปลงชั่วคราวเพื่อทำงานอื่นก่อน

สรุป

Git เป็นเครื่องมือที่ทรงพลังสำหรับการพัฒนาโค้ดในทีม ใช้งานง่ายเมื่อเข้าใจแนวคิดพื้นฐาน และช่วยให้การจัดการโครงการขนาดใหญ่เป็นระบบมากขึ้น

1.1.2. พื้นฐานของ Pointer และการใช้งานใน C++

1.1.2.1. Pointer คืออะไร

- 1) Pointer คือ ตัวแปรที่ใช้เก็บ ตำแหน่งหน่วยความจำ (memory address) ของตัวแปรหรือข้อมูลอื่น
- 2) Pointer ทำให้เราสามารถ:
 - a) เข้าถึงข้อมูลในหน่วยความจำได้โดยตรง
 - b) ส่งค่าระหว่างฟังก์ชันโดยไม่ต้องคัดลอกข้อมูล (pass-by-reference)
 - c) จัดการหน่วยความจำแบบไดนามิก (dynamic memory)

1.1.2.2. การประกาศ Pointer

Syntax:

```
<data_type>* <pointer_name>;
```

ตัวอย่าง:

```
int x = 10;           // ตัวแปรปกติ
int* ptr = &x;        // ptr เก็บตำแหน่งของ x
```

คำอธิบาย:

- & : เครื่องหมายอ้างถึง "ตำแหน่งหน่วยความจำ" (address-of operator)
- * : ใช้เพื่อ "เข้าถึงค่าที่ตำแหน่งนั้น" (dereference operator)

1.1.3.การใช้งาน Pointer

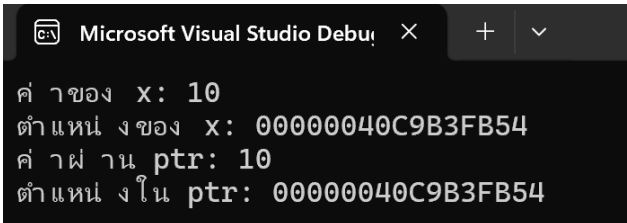
1.1.3.1. การเข้าถึงตำแหน่งและค่าผ่าน Pointer

```
#include <iostream>
using namespace std;

int main() {
    int x = 10;
    int* ptr = &x; // เก็บ address ของ x

    cout << "ค่าของ x: " << x << endl;           // แสดงค่าของ x
    cout << "ตำแหน่งของ x: " << &x << endl;       // แสดง address ของ x
    cout << "ค่าผ่าน ptr: " << *ptr << endl;      // เข้าถึงค่าผ่าน ptr
    cout << "ตำแหน่งใน ptr: " << ptr << endl;    // แสดง address ที่ ptr เก็บไว้

    return 0;
}
```



Microsoft Visual Studio Debug Console output:

```
ค่าของ x: 10
ตำแหน่งของ x: 00000040C9B3FB54
ค่าผ่าน ptr: 10
ตำแหน่งใน ptr: 00000040C9B3FB54
```

1.1.3.2. การเปลี่ยนค่าผ่าน Pointer

ใช้ Pointer เพื่อเปลี่ยนค่าของตัวแปรโดยตรง:

```
#include <iostream>
using namespace std;

void changeValue(int* ptr) {
    *ptr = 20; // เปลี่ยนค่าของตัวแปรที่ ptr จี้
}

int main() {
    int x = 10;
    cout << "ก่อนเปลี่ยนค่า: " << x << endl;

    changeValue(&x); // ส่ง address ของ x เข้าไป
    cout << "หลังเปลี่ยนค่า: " << x << endl;

    return 0;
}
```

การเปรียบเทียบ Pointer กับการคัดลอกข้อมูล

การคัดลอกข้อมูล (Pass-by-Value)

ข้อดี

1. ความปลอดภัย:
 - การเปลี่ยนแปลงค่าจะไม่ส่งผลต่อค่าต้นฉบับ เพราะเป็นการส่งสำเนาของข้อมูล
2. ความง่ายในการใช้งาน:
 - เหมาะสำหรับการส่งข้อมูลขนาดเล็ก เช่น int, float
3. ลดข้อผิดพลาดเกี่ยวกับหน่วยความจำ:
 - ไม่มีปัญหาเกี่ยวกับ dangling pointers หรือ memory leaks

ข้อเสีย

1. เปลืองหน่วยความจำ:
 - ข้อมูลที่ถูกคัดลอกต้องการพื้นที่เพิ่มขึ้น
 - ไม่เหมาะสำหรับข้อมูลขนาดใหญ่ เช่น อาร์เรย์หรือโครงสร้างข้อมูล
2. ประสิทธิภาพต่ำ:
 - การคัดลอกข้อมูลขนาดใหญ่อาจใช้เวลาและทรัพยากรมาก

```
#include <iostream>
using namespace std;

void modifyValue(int x) {
    x = 20; // การเปลี่ยนค่าไม่มีผลกับตัวแปรต้นฉบับ
}

int main() {
    int a = 10;
    modifyValue(a);
    cout << "ค่าของ a: " << a << endl; // ยังคงเป็น 10
    return 0;
}
```

1.1.3.3. *Pointer Arithmetic*

Pointer สามารถใช้การคำนวณเพื่อเข้าถึงตำแหน่งในอาร์เรย์:

```
#include <iostream>
using namespace std;

int main() {
    int arr[3] = { 10, 20, 30 };
    int* ptr = arr; // ชี้ไปที่ตำแหน่งเริ่มต้นของ arr

    cout << "ค่าใน arr[0]: " << *ptr << endl;
    cout << "ค่าใน arr[1]: " << *(ptr + 1) << endl;
    cout << "ค่าใน arr[2]: " << *(ptr + 2) << endl;

    return 0;
}
```

1.1.3.4. *Dynamic Memory Allocation*

ใช้จัดการหน่วยความจำใน runtime:

- 1) การใช้ new สร้างหน่วยความจำ
- 2) การใช้ delete ลบหน่วยความจำ

```
#include <iostream>
using namespace std;

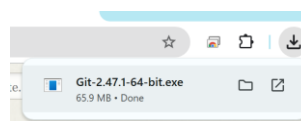
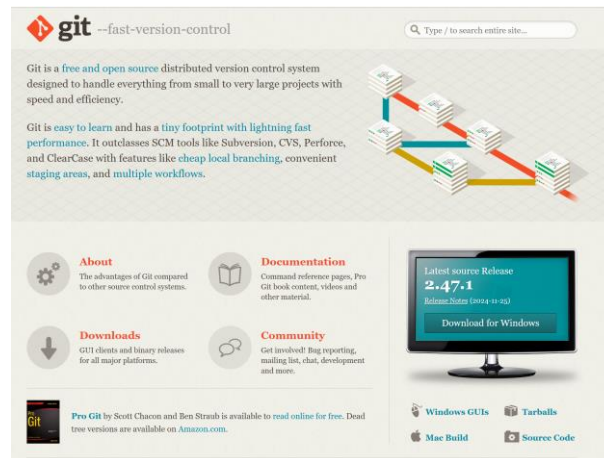
int main() {
    int* ptr = new int; // สร้างหน่วยความจำสำหรับ int
    *ptr = 100;          // กำหนดค่าให้กับตัวแปรในหน่วยความจำ
    cout << "ค่าใน ptr: " << *ptr << endl;

    delete ptr;          // ลบหน่วยความจำ
    return 0;
}
```

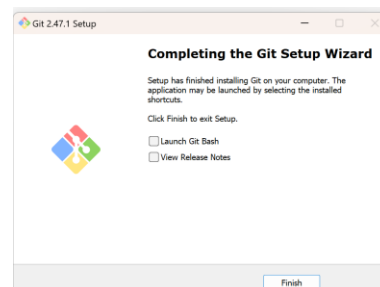
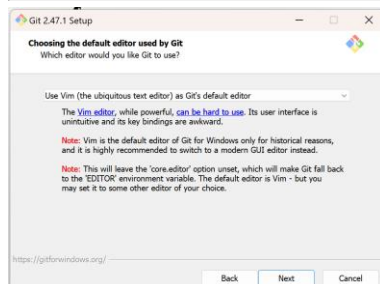
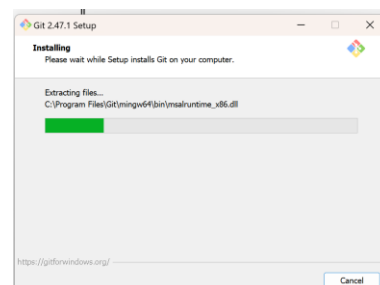
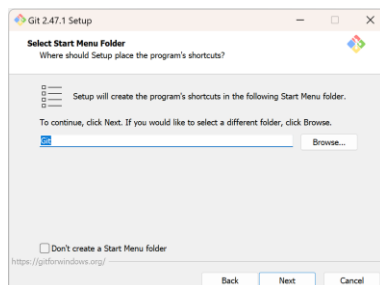
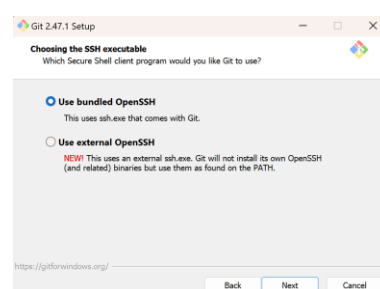
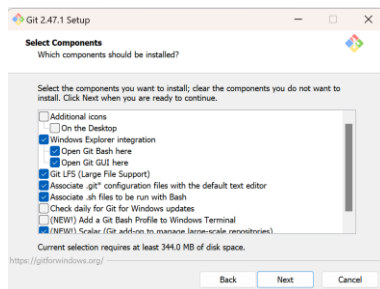
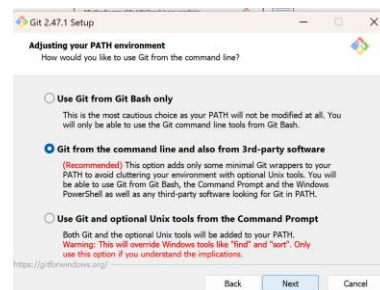
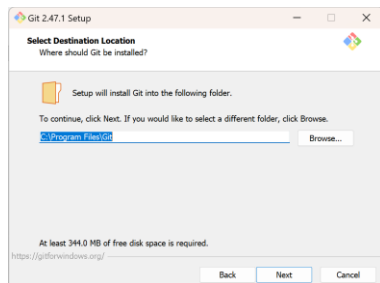
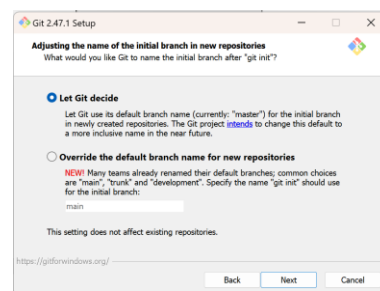

1.2. ปฏิบัติ (3 ชม.)

1.2.1. ติดตั้ง Git

1) ดาวน์โหลด Git จากเว็บไซต์ <https://git-scm.com/>

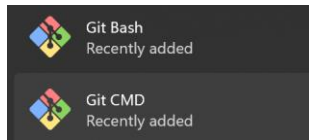


2) ติดตั้งตามขั้นตอน และตรวจสอบว่า Git ติดตั้งเรียบร้อยแล้ว:



git --version

ไปที่ Git CMD

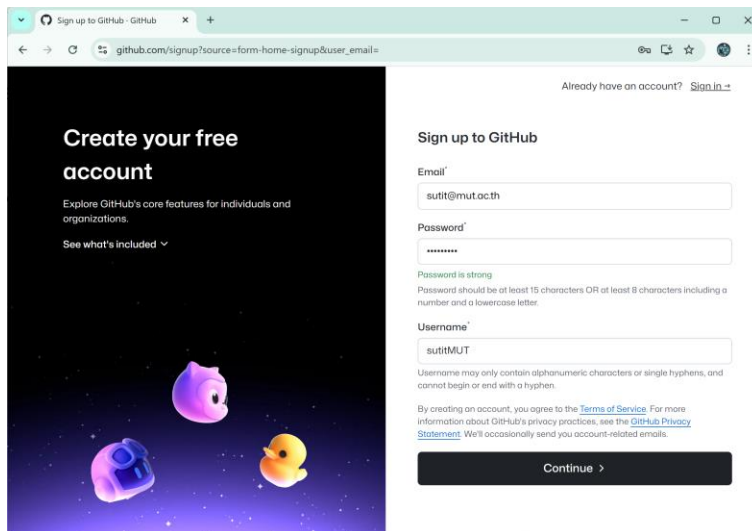
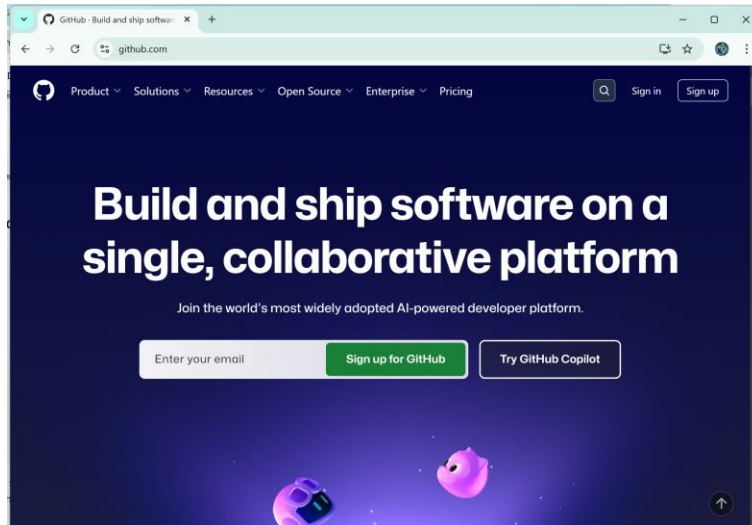
A screenshot of a terminal window titled 'Git CMD'. The prompt is 'C:\Users\Maori>'. The command entered is 'git --version', and the output is 'git version 2.47.1.windows.1'. The prompt is now 'C:\Users\Maori>'.

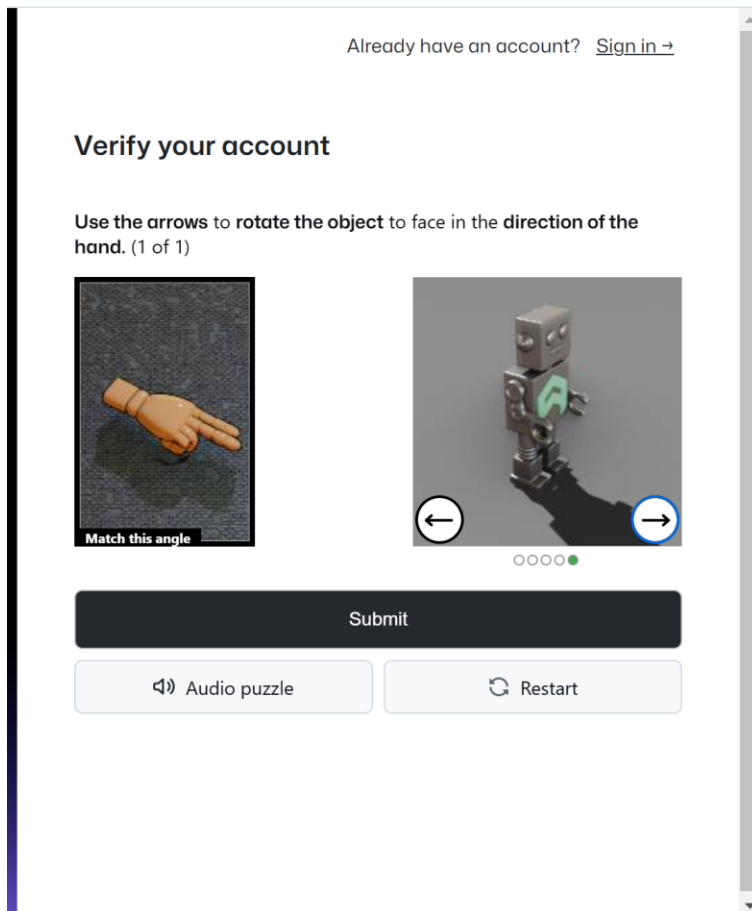
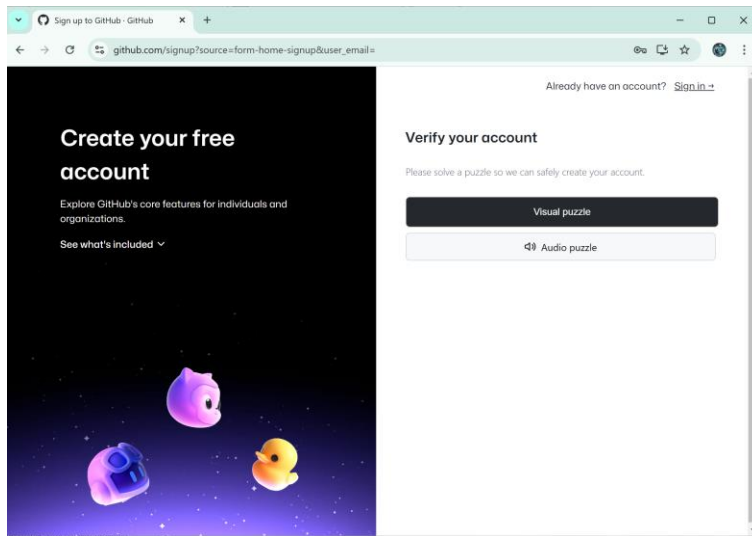
```
C:\Users\Maori>git --version
git version 2.47.1.windows.1
C:\Users\Maori>
```

ให้นักศึกษา Capture ผลการรัน

1.2.2. สมัครใช้งาน เข้าสู่เว็บไซต์ GitHub

ไปที่เว็บไซต์ [GitHub](https://github.com)







Sign in to GitHub

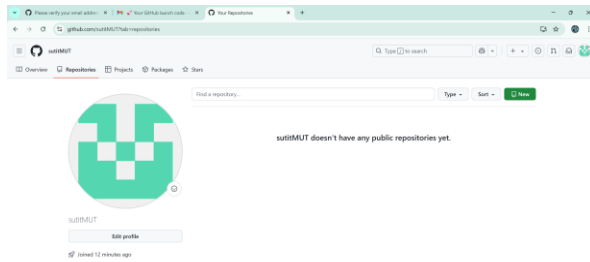
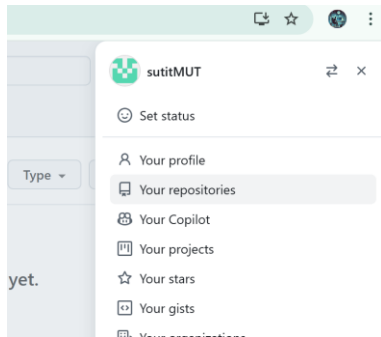
Your account was created successfully. Please sign in to continue

Username or email address

Password [Forgot password?](#)

[Sign in](#)

[Sign in with a passkey](#)
New to GitHub? [Create an account](#)



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

sutitMUT

Repository name *

MIIA0106

✓ MIIA0106 is available.

Great repository names are short and memorable. Need inspiration? How about [friendly-octo-invention](#) ?

Description (optional)

Python and C Programming Language (2/2024)

☐ Public

Anyone on the internet can see this repository. You choose who can commit.

☒ Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

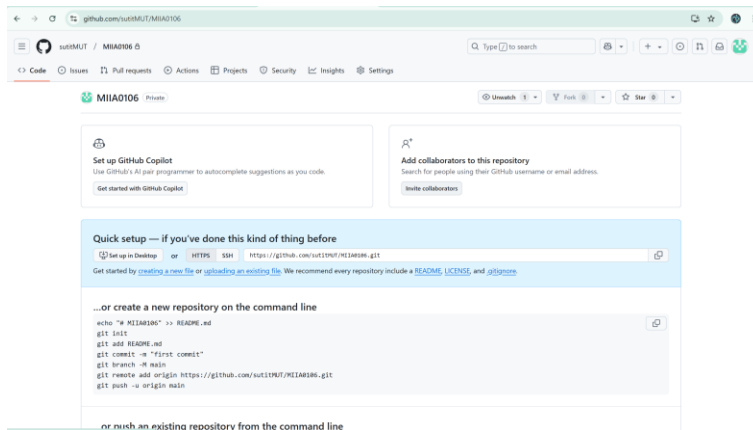
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a private repository in your personal account.

Create repository

ให้นักศึกษา Capture การตั้งค่า



1.2.3.GitHub กับ Sourcetree

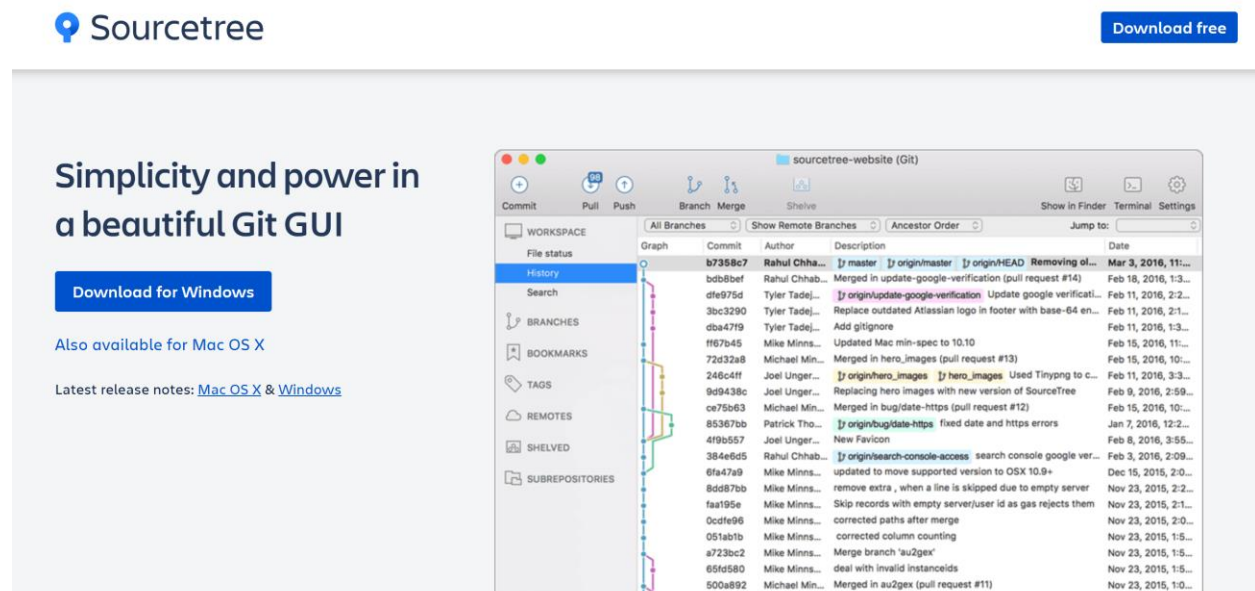
GitHub กับ Sourcetree เป็นเครื่องมือที่ใช้ร่วมกันเพื่อจัดการและควบคุมการเปลี่ยนแปลงในโค้ดได้ง่ายขึ้น โดย GitHub เป็นแพลตฟอร์มสำหรับจัดเก็บ Repository และ Sourcetree เป็นโปรแกรมที่ช่วยจัดการ Git ผ่าน GUI (Graphical User Interface) ไม่ต้องใช้คำสั่งใน Command Line

การติดตั้งและใช้งาน GitHub กับ Sourcetree

1. ติดตั้ง Sourcetree

1. ดาวน์โหลด Sourcetree:

- o [Sourcetree Download](#)



2. ติดตั้ง Sourcetree ตามขั้นตอนในระบบปฏิบัติการของคุณ (Windows/MacOS)

2. สมัครหรือเข้าสู่ระบบ GitHub

1. หากยังไม่มีบัญชี GitHub ให้สมัครที่ [GitHub Signup](#)
2. หากมีบัญชีแล้ว ให้เข้าสู่ระบบเพื่อเตรียมใช้งาน Sourcetree กับ GitHub

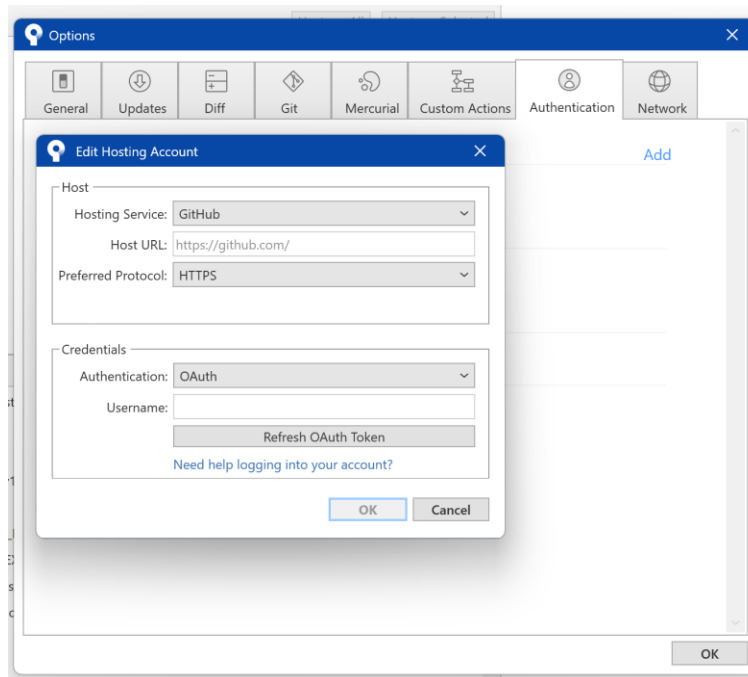
3. เชื่อม Sourcetree กับ GitHub


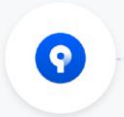
1. เปิด Sourcetree แล้วไปที่:

- Tools > Options (Windows) หรือ Sourcetree > Preferences (Mac)


2. ในแท็บ Authentication:


- คลิก Add เพื่อเพิ่มบัญชี GitHub
- เลือก OAuth เพื่อเชื่อมบัญชี GitHub กับ Sourcetree
- ล็อกอินบัญชี GitHub และยอมรับการเชื่อมต่อ








Authorize SourcetreeForWindows


**SourcetreeForWindows** by [Atlassian](#)
wants to access your GitHub account


**GPG Keys**
Full control of your GPG keys.


**Organizations and teams**
Admin access


**Organization webhooks**
Admin access

**Public SSH keys**
Admin access










**Repository webhooks and services**
Admin access

**Delete repositories**
Ability to delete any adminable repository

**Gists**
Read and write access

**Notifications**
Read access

Full control of your GPG keys.

-  **Organizations and teams**
Admin access
-  **Organization webhooks**
Admin access
-  **Public SSH keys**
Admin access
-  **Repository webhooks and services**
Admin access
-  **Delete repositories**
Ability to delete any adminable repository
-  **Gists**
Read and write access
-  **Notifications**
Read access
-  **Repositories**
Public and private
-  **Personal user data**
Full access

Authorizing will redirect to
<http://localhost:34106>

Options

Edit Hosting Account

Host

Hosting Service: GitHub

Host URL: <https://github.com/>

Preferred Protocol: HTTPS

Credentials

Authentication: OAuth

Username: sutimUT

[Need help logging into your account?](#)

☒ Authentication OK

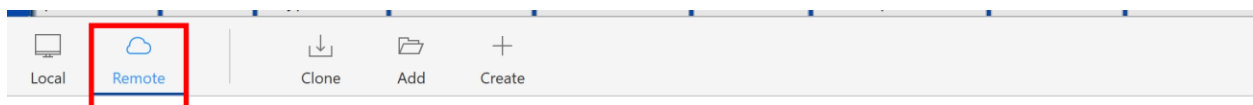
4. สร้างหรือ Clone Repository จาก GitHub

กรณีสร้าง Repository ใหม่ใน GitHub

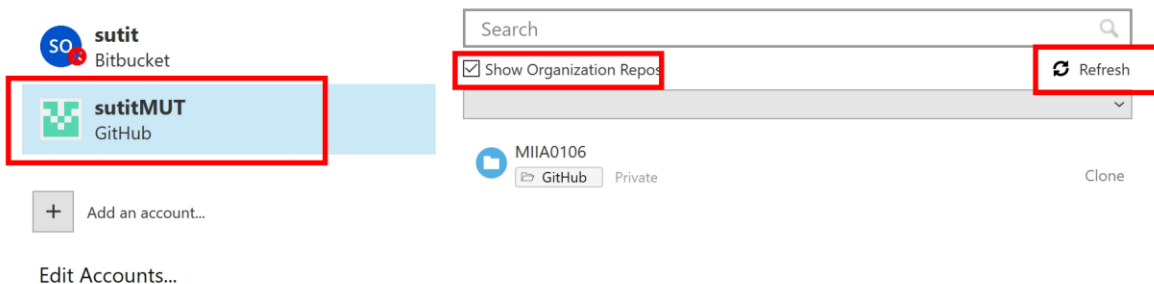
1. ไปที่ GitHub และสร้าง Repository ใหม่:
 - กด **New Repository** และตั้งชื่อ เช่น MyProject
 - กด **Create Repository**
2. คัดลอก URL ของ Repository (HTTPS หรือ SSH)
3. กลับมาที่ Sourcetree:
 - คลิก **Clone Repository**
 - วาง URL ของ Repository ในช่อง **Source Path/URL**
 - เลือกตำแหน่งในเครื่องที่ต้องการเก็บโปรเจกต์ แล้วคลิก **Clone**

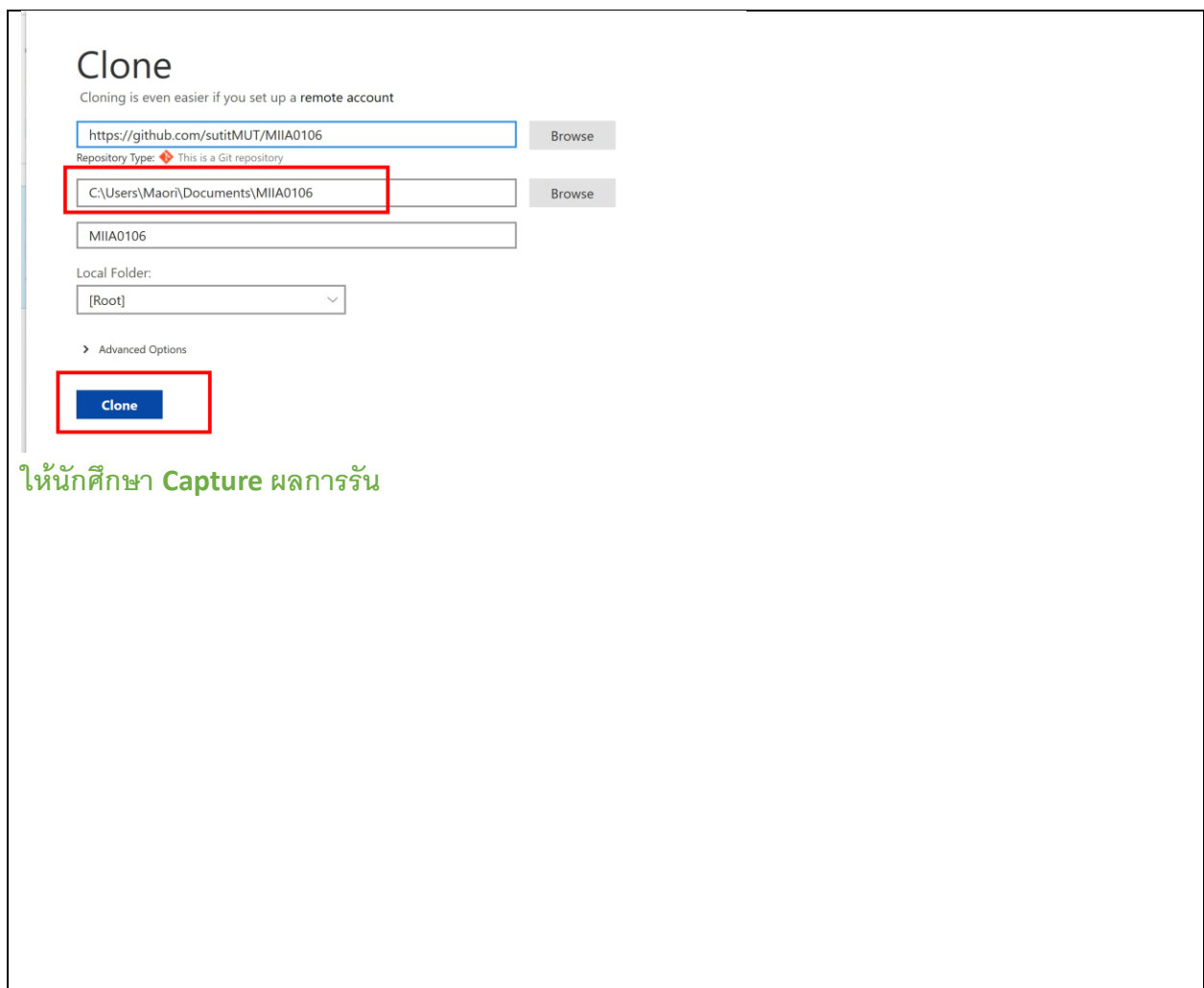
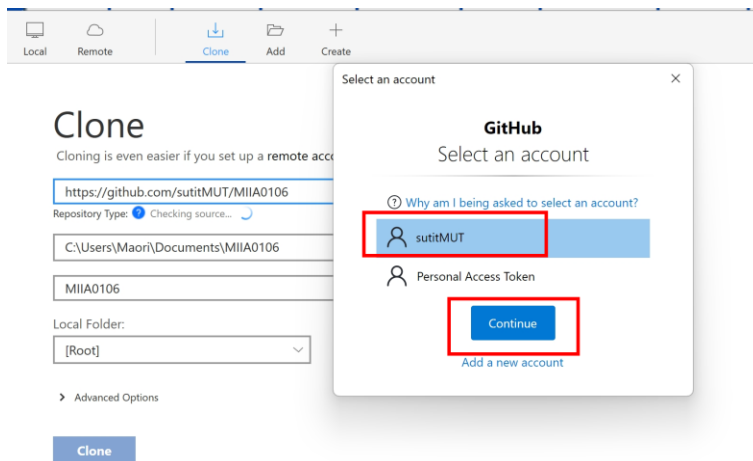
กรณี Clone Repository ที่มีอยู่แล้ว

1. คัดลอก URL จาก Repository บน GitHub
2. ใน Sourcetree:
 - คลิก **Clone Repository**
 - วาง URL และเลือกตำแหน่งในเครื่อง จากนั้นกด **Clone**

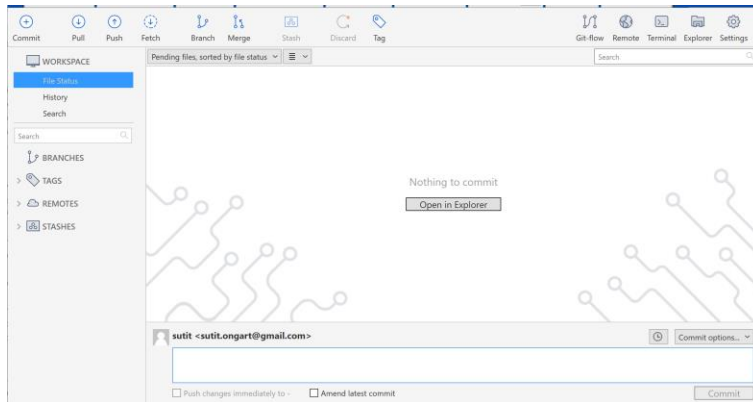


Remote repositories

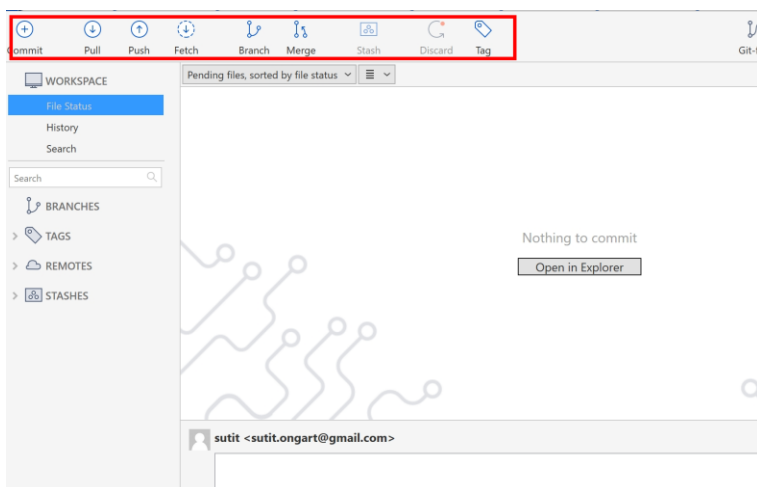




ให้นักศึกษา Capture ผลการรัน



5. การใช้งาน Sourcetree



5.1 ตรวจสอบสถานะโค้ด

- ดูไฟล์ที่เปลี่ยนแปลง (Modified Files) และไฟล์ใหม่ (Untracked Files) ในหน้าหลักของ Sourcetree

5.2 การเพิ่มไฟล์ไปยัง Staging Area

- คลิกเลือกไฟล์ที่ต้องการเพิ่ม แล้วกด **Stage**

5.3 Commit การเปลี่ยนแปลง

- ใส่ข้อความ Commit ในช่องข้อความ (เช่น "Add README.md")
- กด **Commit** เพื่อบันทึกการเปลี่ยนแปลงในเครื่อง

5.4 Push การเปลี่ยนแปลงไปยัง GitHub

- คลิกปุ่ม **Push** และเลือก Branch ที่ต้องการส่งการเปลี่ยนแปลงไปยัง GitHub

5.5 Pull การเปลี่ยนแปลงจาก GitHub

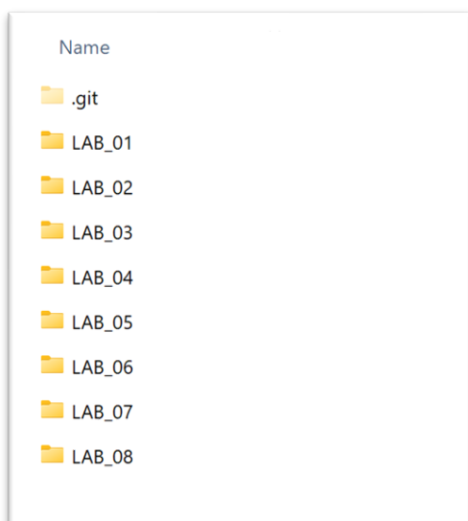
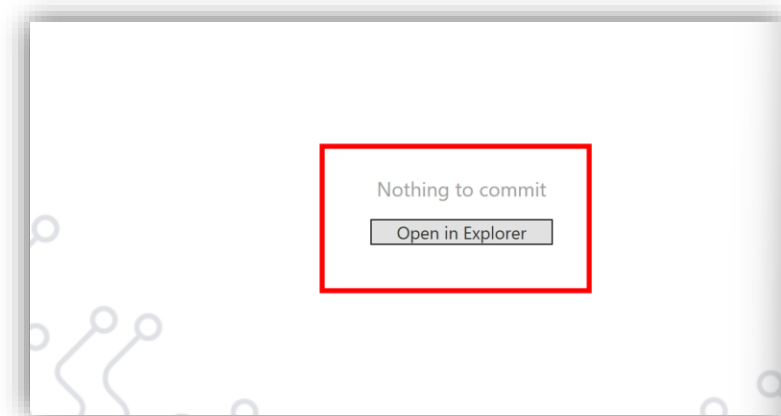
- คลิกปุ่ม **Pull** เพื่อดึงโค้ดใหม่จาก GitHub มายังเครื่องของคุณ

5.6 การสร้าง Branch

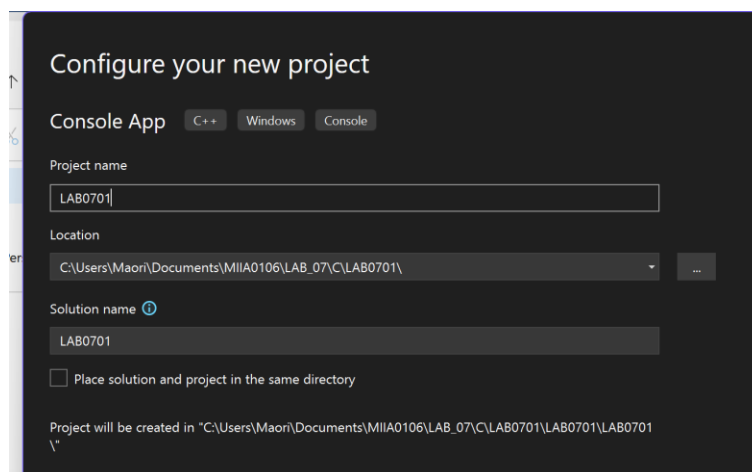
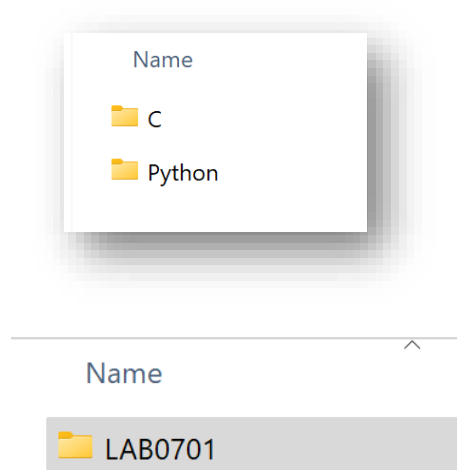
- คลิกปุ่ม **Branch** เพื่อสร้าง Branch ใหม่
- ตั้งชื่อ Branch เช่น feature/login แล้วกด **Create Branch**

5.7 การ Merge Branch

- คลิกที่ Branch หลัก (เช่น main) และกด **Merge** เพื่อรวมการเปลี่ยนแปลงจาก Branch อื่น



ให้นักศึกษา Capture ผลการรัน

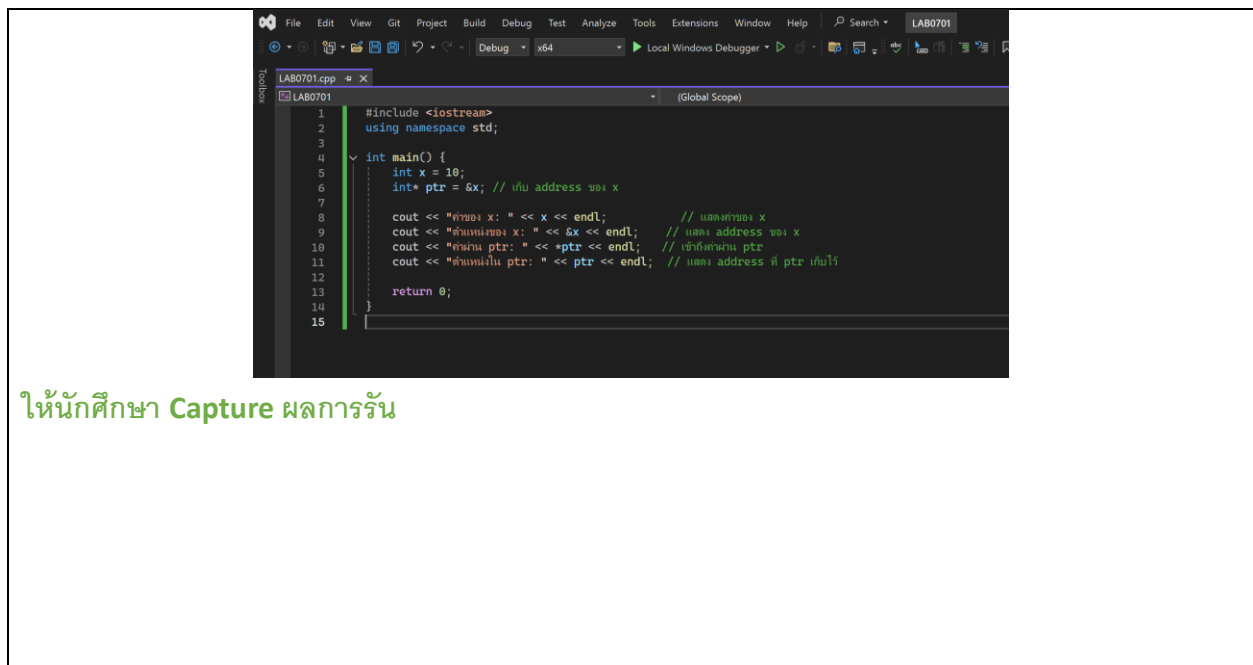


```
#include <iostream>
using namespace std;

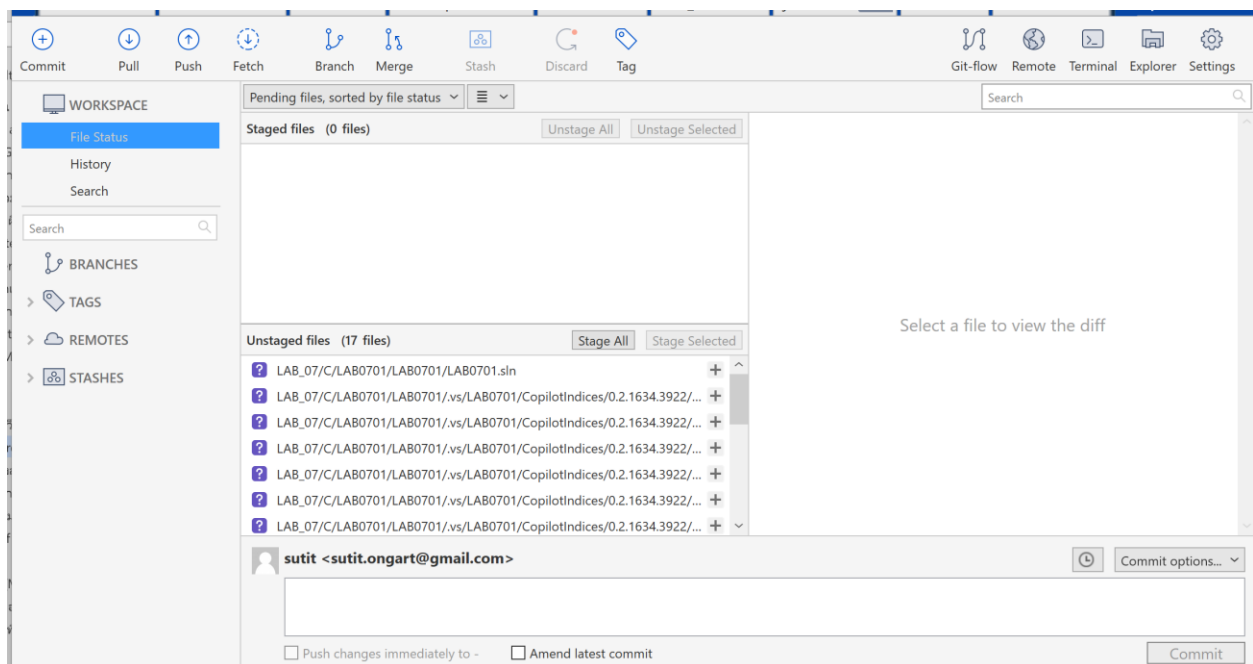
int main() {
    int x = 10;
    int* ptr = &x; // เก็บ address ของ x

    cout << "ค่าของ x: " << x << endl;           // แสดงค่าของ x
    cout << "ตำแหน่งของ x: " << &x << endl;       // แสดง address ของ x
    cout << "ค่าผ่าน ptr: " << *ptr << endl;       // เข้าถึงค่าผ่าน ptr
    cout << "ตำแหน่งใน ptr: " << ptr << endl;     // แสดง address ที่ ptr เก็บไว้

    return 0;
}
```



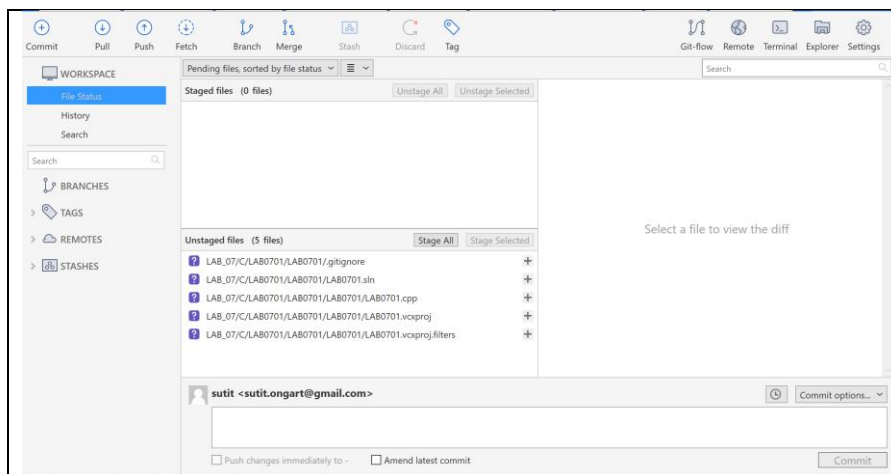
ในโปรเจกต์ C++ การใช้ไฟล์ .gitignore เป็นสิ่งสำคัญเพื่อหลีกเลี่ยงการติดตามไฟล์ที่ไม่จำเป็น เช่น ไฟล์ที่สร้างโดยคอมไพเลอร์ (.o, .exe), ไฟล์สำรอง, หรือไฟล์เฉพาะเครื่อง เช่นการตั้งค่า IDE



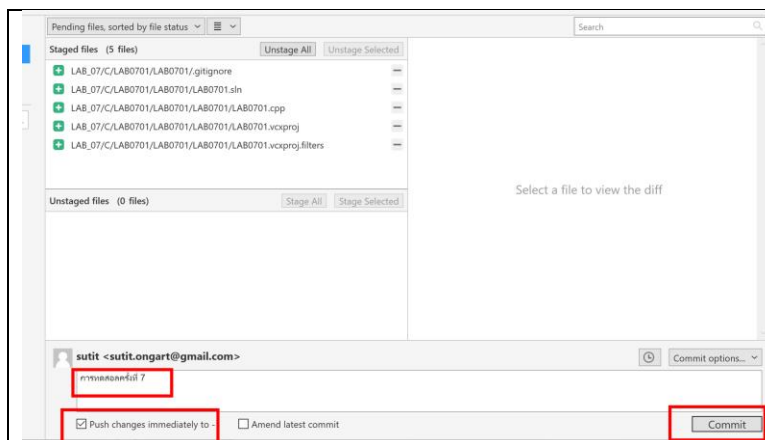
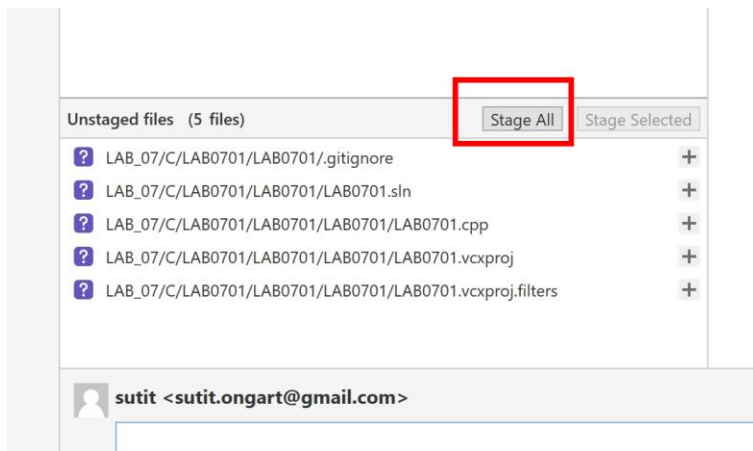
dotnet new gitignore

```
Developer PowerShell
+ Di [Icons]
*****
** Visual Studio 2022 Developer PowerShell v17.11.2
** Copyright (c) 2022 Microsoft Corporation
*****
PS C:\Users\Maori\Documents\MIIA0106\LAB_07\C\LAB0701\LAB0701> dotnet new gitignore
The template "dotnet gitignore file" was created successfully.

PS C:\Users\Maori\Documents\MIIA0106\LAB_07\C\LAB0701\LAB0701> 
```



ให้นักศึกษา Capture ผลการรัน



ให้นักศึกษา Capture ผลการรัน

The top image shows the Visual Studio interface with the 'Commit' window open. It displays a commit message 'การทดลองครั้งที่ 7' and a list of files to be committed, including 'LAB_07/C/LAB0701/LAB0701.gitignore', 'LAB_07/C/LAB0701/LAB0701.sln', 'LAB_07/C/LAB0701/LAB0701/LAB0701.cpp', 'LAB_07/C/LAB0701/LAB0701/LAB0701.vcxproj', and 'LAB_07/C/LAB0701/LAB0701/LAB0701.vcxproj.filters'. The bottom image shows the GitHub web interface for the repository 'sutitMUT / MIIA0106'. It displays the commit history for the 'main' branch, showing a commit by 'sutit' with the message 'การทดลองครั้งที่ 7'.

ให้นักศึกษา Capture ผลการรัน

แก้ไข Code ตามนี้

```
#include <iostream>
using namespace std;

int main() {
    int x = 10;
    int* ptr = &x; // เก็บ address ของ x

    cout << "ค่าของ x: " << x << endl; // แสดงค่าของ x
```

```

cout << "ตำแหน่งของ x: " << &x << endl; // แสดง address ของ x
cout << "ค่าผ่าน ptr: " << *ptr << endl; // เข้าถึงค่าผ่าน ptr
cout << "ตำแหน่งใน ptr: " << ptr << endl; // แสดง address ที่ ptr เก็บไว้

int y = 20;
ptr = &y; // เก็บ address ของ x

cout << "ค่าของ y: " << x << endl; // แสดงค่าของ x
cout << "ตำแหน่งของ y: " << &x << endl; // แสดง address ของ x
cout << "ค่าผ่าน ptr: " << *ptr << endl; // เข้าถึงค่าผ่าน ptr
cout << "ตำแหน่งใน ptr: " << ptr << endl; // แสดง address ที่ ptr เก็บไว้

return 0;
}

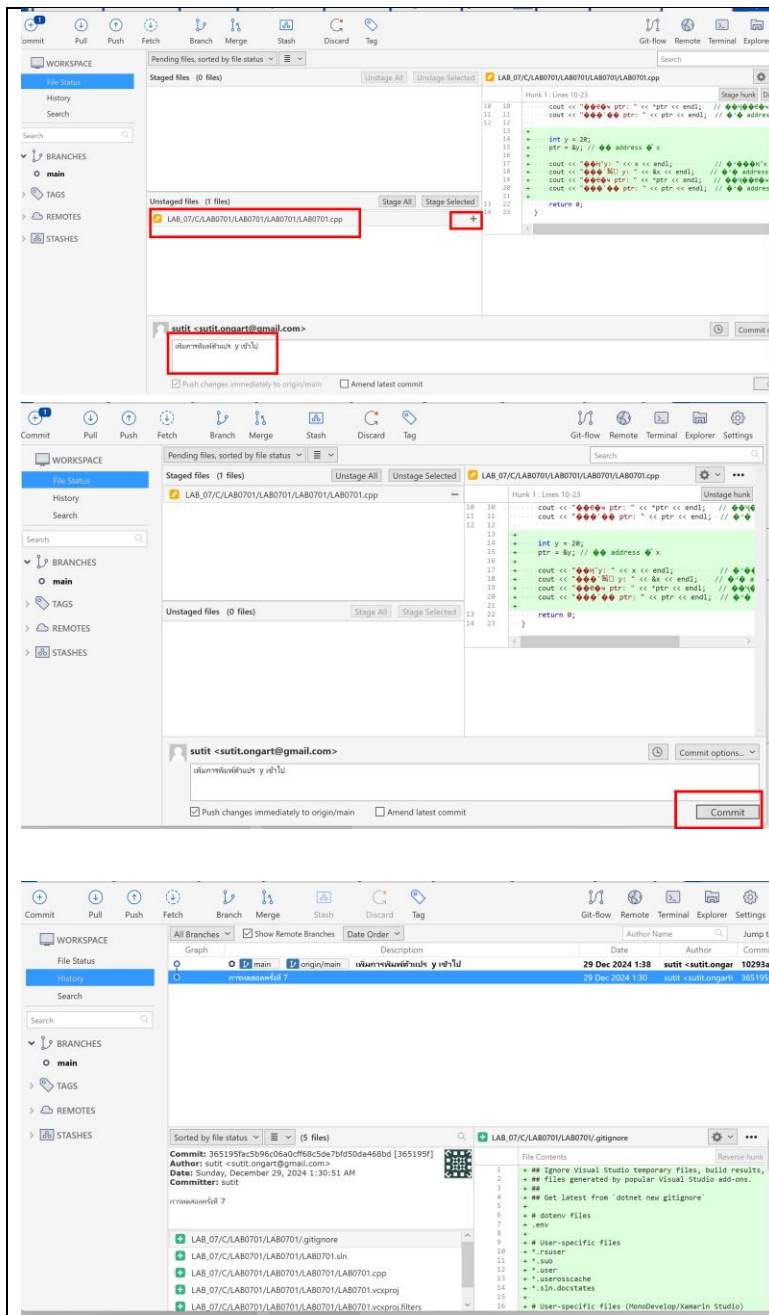
```

```

LAB0701.cpp
LAB0701 (Global Scope)
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int x = 10;
6     int* ptr = &x; // เก็บ address ของ x
7
8     cout << "ค่าของ x: " << x << endl; // แสดงค่าของ x
9     cout << "ตำแหน่งของ x: " << &x << endl; // แสดง address ของ x
10    cout << "ค่าผ่าน ptr: " << *ptr << endl; // เข้าถึงค่าผ่าน ptr
11    cout << "ตำแหน่งใน ptr: " << ptr << endl; // แสดง address ที่ ptr เก็บไว้
12
13
14    int y = 20;
15    ptr = &y; // เก็บ address ของ x
16
17    cout << "ค่าของ y: " << x << endl; // แสดงค่าของ x
18    cout << "ตำแหน่งของ y: " << &x << endl; // แสดง address ของ x
19    cout << "ค่าผ่าน ptr: " << *ptr << endl; // เข้าถึงค่าผ่าน ptr
20    cout << "ตำแหน่งใน ptr: " << ptr << endl; // แสดง address ที่ ptr เก็บไว้
21
22    return 0;
23 }
24

```

ให้นักศึกษา Capture ผลการรัน



ให้นักศึกษา Capture ผลการรัน

1.2.4.เขียนโปรแกรมเรียงลำดับข้อมูลในอาร์เรย์

1.2.5.ใช้พอยน์เตอร์ในการสลับค่าของตัวแปรสองตัว

1.2.6.สร้างฟังก์ชันหาค่ามากที่สุดและน้อยที่สุดในอาร์เรย์

